הטכניון – מכון טכנולוגי לישראל

מעבדה במערכות הפעלה 046210

תרגיל בית מס' 3

תאריך הגשה: 25 באוגוסט 2024 עד השעה 23:55

הקדמה

ברוכים הבאים לתרגיל מס' 3 במעבדה במערכות הפעלה. במסגרת תרגיל זה נבנה מנהל התקן פשוט שתומך בפעולות קריאה, כתיבה ובמנגנון ioctl (ראו הסבר על מנגנון זה בהמשך). מנהל ההתקן יהיה מסוג Char Device אשר יטען כ-LKM.

תיאור כללי

מטרתכם היא לייצר התקן עבור תקשורת מוצפנת בין תהליכים. תהליכים יכתבו הודעה להתקן, שיצפין אותה וישמור אותה בחוצץ פנימי. לאחר מכן תהליך אחר יוכל לקרוא מהתקן, כדי לקבל את ההודעה לאחר פיענוח.

ההצפנה תעשה באמצעות צופן ויז'נר. זהו צופן החלפה, המחליף כל אות במסר באות אחרת על ידי הזזה (מחזורית) על פי מפתח. השימוש במפתח נעשה באופן מחזורי. לדוגמה, נרצה להצפין את המשפט: "Solution to HW3" בעזרת המפתח 12345678. פעולת ההצפנה תתבצע כדלקמן:

Plain text	S	o	l	u	t	i	o	n		t	o		H	W	3
Key	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7
Cipher text	T	q	o	y	y	o	v	v		v	r		M	c	A

נשים לב כי בתהליך ההצפנה :

- האותיות מוחלפות על ידי הזזה לאות שמספרה הסידורי באלפבית המורחב (ראו להלן) הוא סכום המספר הסידורי של האות המקורית והמפתח הנוכחי. לדוגמה: S היא האות ה-19 באלפבית, והמפתח המתאים הוא 1. לכן היא מוחלפת באות T שהיא האות ה-20 באלפבית המורחב.
- האלפבית המורחב מורכב משרשור האותיות הלועזיות הגדולות (uppercase) , האותיות הלועזיות הקטנות (lowercase) והספרות. כלומר משמאל לימין:

  ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789

  סה"כ באלפבית המורחב ישנם 62 תווים.

- המפתח חוזר על עצמו באופן מחזורי, כשהתו הראשון הוא התו השמאלי ביותר.
- רק אותיות וספרות יעברו הצפנה. תווים אחרים (כל תו אחר שלא שייך לאלפבית המורחב, כגון רווח בדוגמה) לא מוצפנים אך מקדמים את המפתח.
- פעולת ההזזה הינה מחזורית במספר התווים באלפבית המורחב

כאמור, המשפט: "Solution to HW3" יוצפן למשפט "Tqoyyovv vr McA". פעולת הפיענוח מתבצעת בצורה זהה, רק שבמקום חיבור יתבצע חיסור, גם פעולת הפענוח היא מחזורית במספר התווים באלפבית המורחב.

פירוט על המימוש

שם ההתקן שתיצרו יהיה vigenere, ועליו לממש את הפעולות הבאות מתוך ממשק הקובץ הרגיל של יוניקס. בסוגריים מופיע שם הפונקציה המתאימה ב-file_operations:

- כתיבה (write) – בכתיבה יש להעתיק את תוכן החוצץ שהועבר (פרמטר buf) לסוף החוצץ הפנימי של ההתקן בצורה מוצפנת (ראו SET_KEY בהמשך). אם אין מקום בחוצץ יש להגדיל אותו בשיטה לבחירתכם (למשל, להקצות חוצץ חדש בגודל החוצץ הישן + גודל ההודעה החדשה ולשחרר את הישן). לאחר מכן יש להחזיר את מספר הבתים שנכתבו (מה ששווה לאורך החוצץ, שמועבר בפרמטר count). במקרה של חוצץ NULL יש להחזיר EFAULT-. במקרה של שגיאה בקריאה מחוצץ המשתמש יש להחזיר EBADF-. במקרה שאין מפתח מוגדר יש להחזיר - EINVAL. במקרה של בעיה בהקצאת זיכרון דינמי יש להחזיר ENOMEM-. בכל שגיאה אחרת יש להחזיר - EFAULT.
- קריאה (read) – כל קריאה תעתיק לחוצץ שהעביר המשתמש (פרמטר buf), לאחר פענוח, את כמות הבתים שהוא ביקש (פרמטר count), או את שארית החוצץ הזמין אם נשארו פחות מ-count בתים. בכל מקרה אין לקרוא מעבר לכמות הבתים שנכתבו, גם אם במימוש שלכם יש מקום בחוצץ אחרי ההודעה האחרונה. ערך ההחזרה יהיה מספר הבתים שנכתבו ל-buf. יש לזכור את המיקום בחוצץ הפנימי של כל תהליך, להתחיל את הקריאה מהמיקום הזה ולקדם אותו בסוף קריאה מוצלחת (ראו פרמטר f_pos). אם אין נתונים לקרוא (כי המשתמש הגיע לסוף החוצץ הזמין) יש להחזיר 0 ולהשאיר את החוצץ של המשתמש ללא שינוי. אם המצביע לחוצץ הוא NULL יש להחזיר -

EFAULT. אם המפתח לא מוגדר יש להחזיר EINVAL-. אם יש בעיה בלהעתיק נתונים למרחב המשתמש יש להחזיר EBADF-. בכל שגיאה אחרת יש להחזיר EFAULT-.

<u>שימו לב על שימוש נכון במפתח ההצפנה:</u> גם בהצפנה וגם בפענוח, מפתח ההצפנה 'מיושר' אל התו הראשון בחוצץ ולא לתו הראשון שנקרא או נכתב.

- חיפוש (llseek) – משנה את מיקום תחילת הקריאה הבאה (f_pos) של התהליך הנוכחי באופן יחסי למיקום הנוכחי, לפי הפרמטר offset. לדוגמה, אם המיקום הנוכחי הוא 100 וה-offset המתקבל הוא 10-, הקריאה הבאה תתחיל ממיקום 90. יש להגביל תזוזה להיות בתוך החוצץ: חיפוש אל מעבר למיקום 0 יזיז לתחילת החוצץ ובאופן דומה חיפוש אל מעבר לסוף החוצץ הזמין (ההודעה האחרונה שנכתבה) יזיז אל סוף החוצץ הזמין. הקריאה תחזיר את המיקום החדש בחוצץ.
- SET_KEY – פעולת ioctl. מקבלת מחרוזת כפרמטר. משנה את המפתח הנוכחי שאיתו יוצפנו ויפוענחו כל ההודעות הבאות. כל תו במחרוזת יומר למספר ע"י מיקומו בא"ב המורחב. לדוגמה, המחרוזת "ABC" תומר למפתח "123". אם המפתח ריק או NULL, יש להחזיר EINVAL-. במקרה של שגיאה בהקצאת זיכרון יש להחזיר ENOMEM-. במקרה של שגיאה בהעתקת המפתח מהמשתמש יש להחזיר EBADF-. במקרה של הצלחה יש להחזיר 0.
- RESET – פעולת ioctl. משחררת את החוצץ הפנימי ואת המפתח.
- DEBUG – פעולת ioctl. מקבלת מספר כפרמטר. אם הוא 1, לא יתבצעו הצפנה או פענוח בקריאות וכתיבות הבאות. אם הוא 0, ההתנהגות כרגיל. אם הפרמטר הוא מספר אחר, מחזירה EINVAL-.

את פעולות SET_KEY, RESET ו-DEBUG יש לממש על ידי מנגנון ה-IOCTL. מספר הפעולה מוגדר עבורכם כקבוע בקובץ שלד. ראו מידע כללי על ioctl בנספח בסוף ודוגמאות מימוש בלינקים למטה.

מספר major של ההתקן צריך להינתן באופן דינמי. קבצי התרגיל (vigenere.h, vigenere.c) מגדירים את הקבועים שהוזכרו בתרגיל ומכילים שלד שעליו ניתן לבנות את מנהל ההתקן. שימו לב שהקבצים שקיבלתם לא בהכרח מכילים שלד לכל הפונקציות שעליכם לממש. היעזרו בחומרי העזר המצוינים למטה לפי הצורך.

דוגמא שימוש

- תהליך א' מבצע SET_KEY ל-"ABC".
- תהליך א' מבצע write למחרוזת "Hello". המחרוזת נכתבת בצורה מוצפנת בחוצץ הפנימי.
- תהליך ב' מבצע read וקורא "Hello". המחרוזת חזרה אחרי פענוח עם אותו המפתח.
- תהליך ב' מבצע llseek(-10) כדי לחזור לתחילת החוצץ.
- תהליך ב' מפעיל DEBUG.
- תהליך ב' מבצע read וקורא "lgomq". המחרוזת לא עברה פענוח בקריאה כי DEBUG פועל.

מידע שימושי

- הסבר על מנהלי התקן מסוג Char Driver + דוגמאות ניתן למצוא בקישור הבא וכן בקישור הבא.
- אפשר למצוא הסבר על איך להוסיף תמיכה ב- IOCTL בקישור הבא.
- ניתן, ורצוי ,לדבג את המודול בעזרת פקודת printk.
- הגרעין לא מקופל אל מול הספריות הסטנדרטיות (GNU libc). לכן לא ניתן לעשות שימוש בפונקציות מתוך 'stdlib.h', 'stdio.h', 'string.h' וכדומה. עם זאת, לרוב הפונקציות השימושיות יש תחליף ומימוש מתאים בגרעין.
- הגרעין עושה שימוש בזיכרון פיזי. לשם הקצאת זיכרון דינמי בגרעין יש לעשות שימוש בפקודות kmalloc ו-kfree אשר מקצות זיכרון באופן דינאמי. שימו לב, עליכם לדאוג לשחרר זיכרון שאתם מקצים, דליפות זיכרון יחשבו כשגיאה.
- זיכרון השייך למרחב הגרעין חייב להיות מופרד מטעמי אבטחה מזיכרון המשתמש. לכן, קוד מנהל ההתקן לא יכול לגשת ישירות למידע הנמצא במרחב הזיכרון של המשתמש באופן ישיר.

בעקבות זאת עליכם לעשות שימוש בשגרות copy_to_user, copy_from_user, strnlen_user על מנת להעביר מידע בין המרחבים. ניתן לקרוא עליהן בקישור הבא ובקישור הבא

בדיקה של מנהל ההתקן

כדי לקמפל את מנהל ההתקן יש להשתמש בפקודה הבאה (לחלופין ניתן להשתמש בקובץ Makefile המצורף):

```
gcc -c -I/usr/src/linux-2.4.18-14/include -Wall vegenere.c
```

לשם בדיקת מנהל ההתקן יש להתקין אותו ולבדוק האם הוא מבצע נכון את כל הפעולות הנדרשות ממנו. טעינת מנהל ההתקן נעשית בעזרת הפקודה:

```
insmod ./vegenere.o
```

כדי ליצור קובץ התקן המשויך למנהל ההתקן ניתן להיעזר בפקודה mknod. לדוגמא, הפקודה הבאה תיצור קובץ התקן בשם vegenere המזוהה על ידי מספר מינורי 0:

```
mknod /dev/vegenere c major 0
```

במקום major יש לרשום את מספר ה-major שנבחר להתקן שלכם. ניתן למצוא את מספר ה-major שנבחר להתקן בקובץ

/proc/devices

הסרת ההתקן נעשית בעזרת הפקודות:

```
rm -f /dev/vegenere
rmmod vegenere
```

מומלץ לכתוב סקריפטים שיבצעו את הפעולות הנ"ל בצורה אוטומטית על מנת לחסוך זמן וטעויות אנוש.

בדיקת תפקוד מנהל ההתקן תתבצע על ידי פתיחת קובץ ההתקן, ביצוע הפעולות המפורטות לעיל ובדיקת התוצאות שלהן. ניתן לבצע זאת בעזרת תכנית בשפת C. כמו כן ניתן גם לעשות זאת בשפת סקריפט כגון Python שמאפשרת פיתוח מהיר ונוח של תכניות. תיעוד שפת Python נמצא בקישור הבא, הסבר על גישה לקבצים נמצא בקישור הבא ועל ביצוע פניות IOCTL ניתן לקרוא בקישור הבא. בין קבצי התרגיל מצורף קובץ בדיקה לדוגמא: test.py.

אין צורך להגיש את התוכנית בה השתמשתם לבדיקת מנהל ההתקן שלכם.

הוראות הגשה לתרגיל:

- הגשה אלקטרונית דרך אתר הקורס.
- יש להגיש דרך חשבון של אחד השותפים בלבד.(אין להגיש פעמיים - מכל חשבון).

ההגשה בקובץ **zip** בשם id1_id2.zip , כאשר id1, id2 הם מספרי ת.ז. של השותפים. הקובץ יכיל:

- קבצי התוכנית: vegenere.h, vegenere.c
- קובץ טקסט submitters.txt עם הנתונים של המגישים לפי המתכונת הבאה:

first_name1 last_name1 id1
first_name2 last_name2 id2

יש להקפיד שקובץ ה zip לא יכיל קבצים נוספים, לרבות תיקיות. דהיינו, על ההגשה להיות בצורה הבאה:

```
zipfile -+
         |
         +- submitters.txt
         +- vegenere.c
         +- vegenere.h
         |
```

דגשים לגבי הציון

- משקל התרגיל הינו 30% מהציון הסופי.
- הקפידו על מילוי הדרישות לשמות הקבצים והממשקים.
- יש להקפיד על סדר ותיעוד הקוד.
- הגשה באיחור תגרור הורדת ניקוד.

נספח: הסבר על מנגנון ה-*ioctl*

בהרבה התקנים (כמו ההתקן שלנו בתרגיל הזה), המנגנון המועדף והפשוט ביותר להעביר מידע בין המשתמש להתקן הוא דרך קובץ שאותו קוראים וכותבים. במנגנון הזה משתמשים כדי לעשות את הפעולות הפשוטות ו\או הפעולות שדורשות להעביר מידע בקצב מהיר. הבעיה היא שמנגנון זה מוגבל לשתי פעולות (קריאה וכתיבה). במקרה וההתקן רוצה לחשוף כלפי המשתמש עוד פונקציות, הוא צריך דרך כלשהי להרחיב את הממשק הקיים. לצורך כך המציאו את מנגנון ioctl – קריאת מערכת כללית שמריצים על קובץ התקן ומעבירים לה מספר פונקציה ופרמטר. מנגנון זה מאפשר להתקן להגדיר פונקציות נוספות, להצמיד להם מספר ולהודיע למשתמש (למשל בתיעוד של ההתקן) מה המשמעות של כל מספר פונקציה ומה הפרמטר שהוא צריך להעביר.

המשתמש לאחר מכן ישתמש בקריאת המערכת ioctl על הקובץ של ההתקן ביחד עם מספר הפונקציה כדי לבצע את הפעולות הנוספות.

דוגמה פשוטה לשימוש ב-ioctl הוא כרטיס קול. ההתקן של כרטיס הקול יצור קובץ התקן, אשר קריאה ממנו תבצע הקלטה (דרך שקע המיקרופון של הכרטיס) וכתיבה אליו תשמיע צליל (דרך שקע הרמקולים). אבל אם המשתמש רוצה לשנות הגדרות של הכרטיס, כמו למשל קצב הדגימה של המיקרופון, הוא יפתח את קובץ ההתקן ויבצע ioctl. במספר הפונקציה הוא יכתוב את מה שכתוב בתיעוד של מנהל ההתקן, ובפרמטר הוא יכול להעביר את קצב הדגימה כמספר חיובי. עבור פונקציה אחרת (של אותו ההתקן או התקן אחר), הפרמטר יכול להיות מספר שלילי או אפילו מצביע. מבחינת ioctl עצמה אין משמעות לפרמטר – המימוש במנהל ההתקן יתייחס לפרמטר לפי הטיפוס המתאים למספר הפונקציה שהמשתמש ביקש להריץ.

שימו לב שמספר הפונקציה הוא שרירותי לחלוטין ופרטי למנהל ההתקן שאתם כותבים. למרות זאת, הוגדרו קונבנציות איך ליצור את המספרים האלה ויש פונקציות עזר לשם כך. זאת המשמעות של שורות ה-define בקובץ השלד – להגדיר את המספרים של פונקציות ה-ioctl בתרגיל לפי הקונבנציות הנהוגות בקרנל.